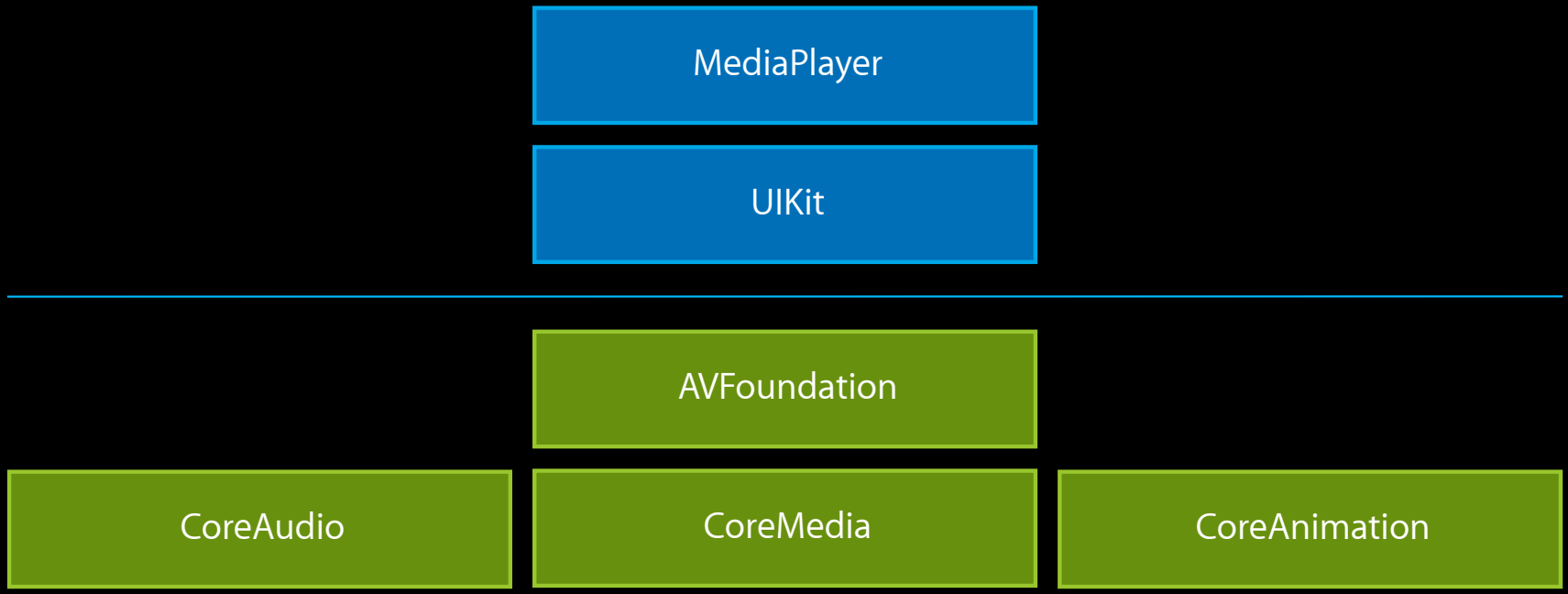
Discovering AV Foundation

Kevin Calhoun
Media Systems Engineering

What You'll Learn

- Why and when you should use AV Foundation
- Concepts underlying representation and manipulation of timed media
- Tasks you can accomplish using AV Foundation

Technology Framework



Using Time-Based Media on iPhone OS

- Several easy ways
  - Playback
    - MPMoviePlayerController and MPMoviePlayerViewController
    - HTML5 video and audio tags
    - For sound files, AVAudioPlayer
  - Capture
    - UIImagePickerController for video recording
    - AVAudioRecorder for audio recording

Why You Should Use AV Foundation 4.0 API

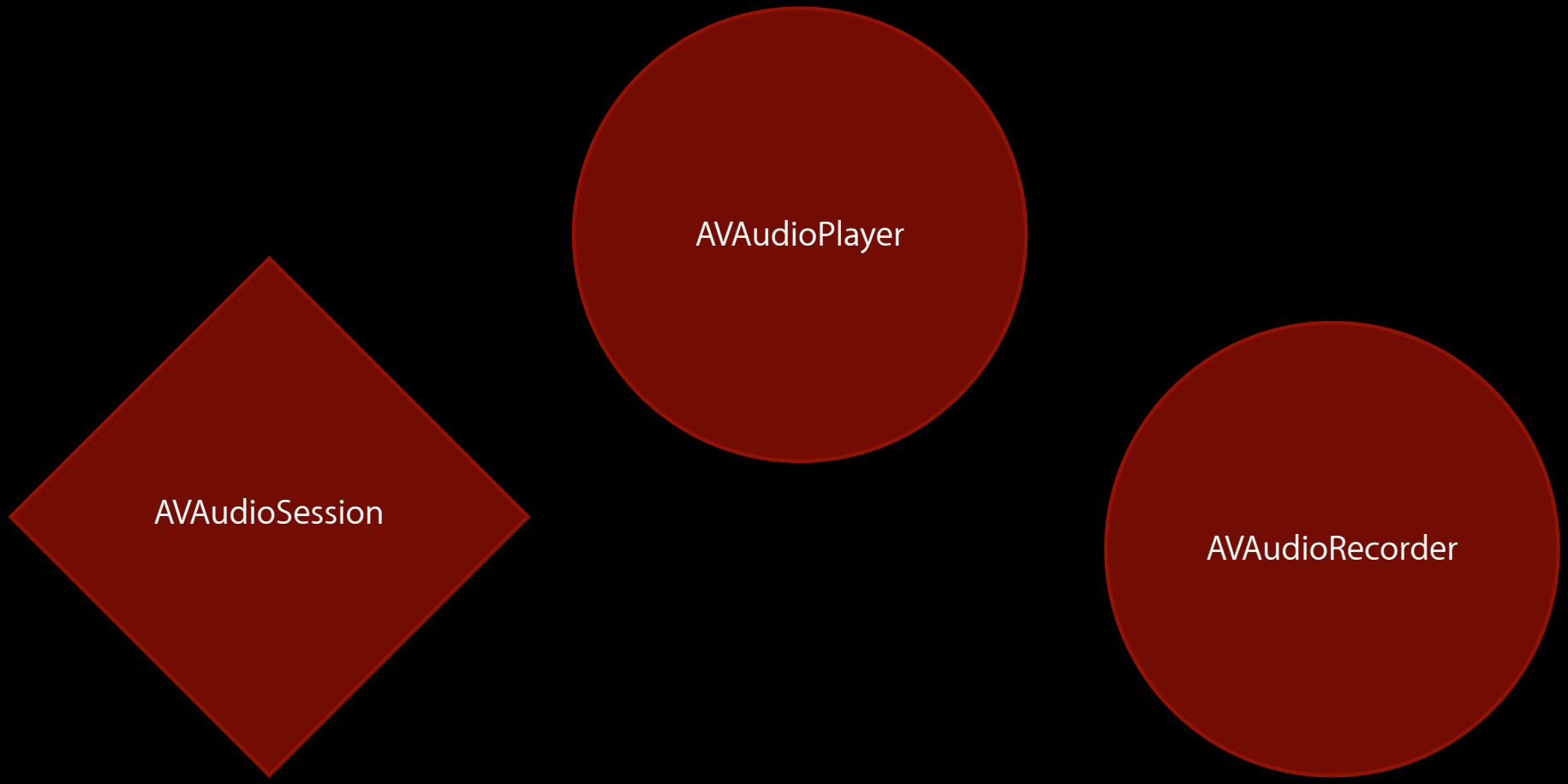
- Inspect
  - If you need to know details of an asset
- Play
  - In more sophisticated ways than are possible with e.g. MPMoviePlayer
- Compose/edit
  - Use the media you want, where and when you want it
- Re-encode/export
  - Create new assets
- Enjoy full access to camera
  - Obtain data from input devices, with the option to write it to storage

Time-based Media Challenges

- Asynchronous operation is natural
  - The content is intended to be processed incrementally
- A summary view of the content isn't always directly available
  - A simple question ("what's the duration?") may require significant work to answer
- Circumstances may change while you work
  - Operations can be interrupted
- The variety of time-based media resources is impressive
  - Uniform treatment of media resources may require work

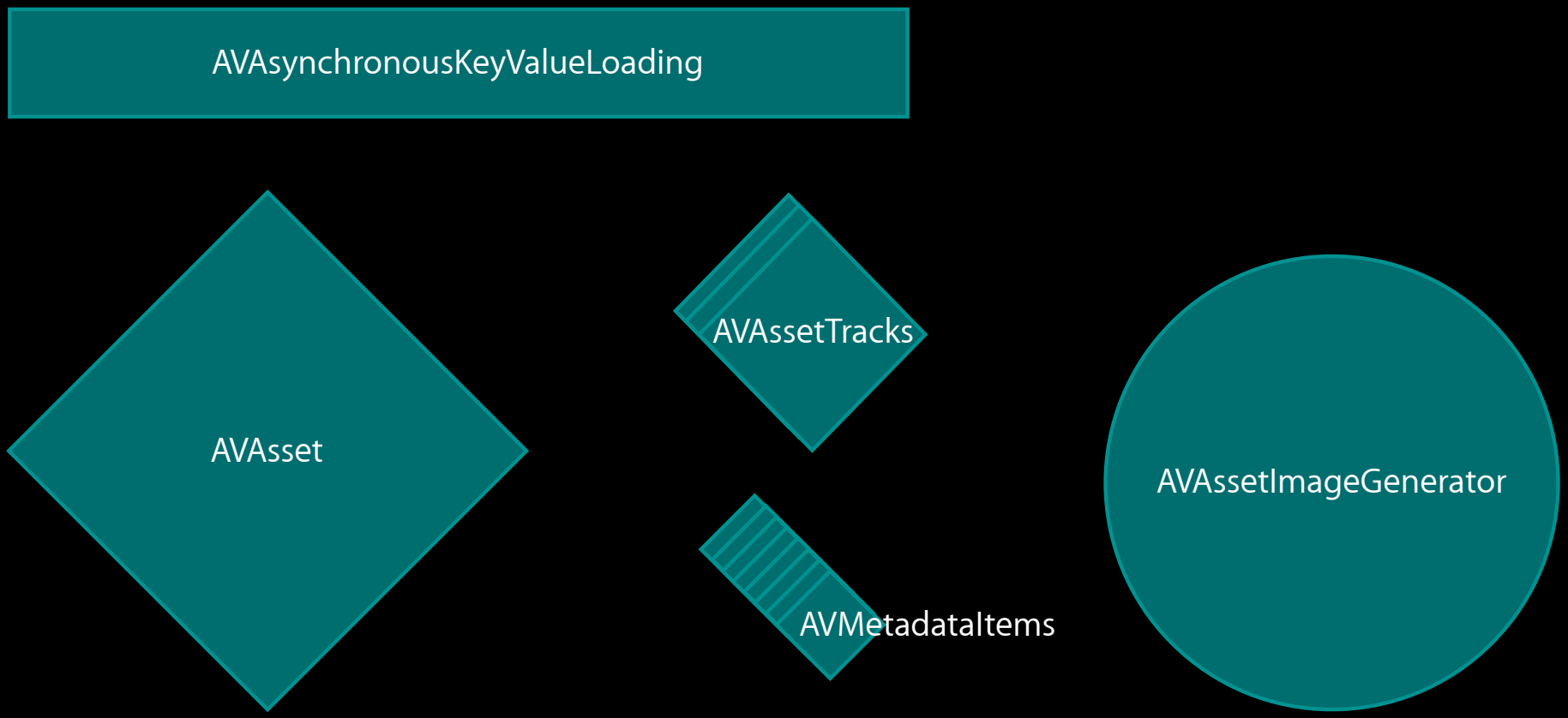
AV Foundation Classes by Function

Audio



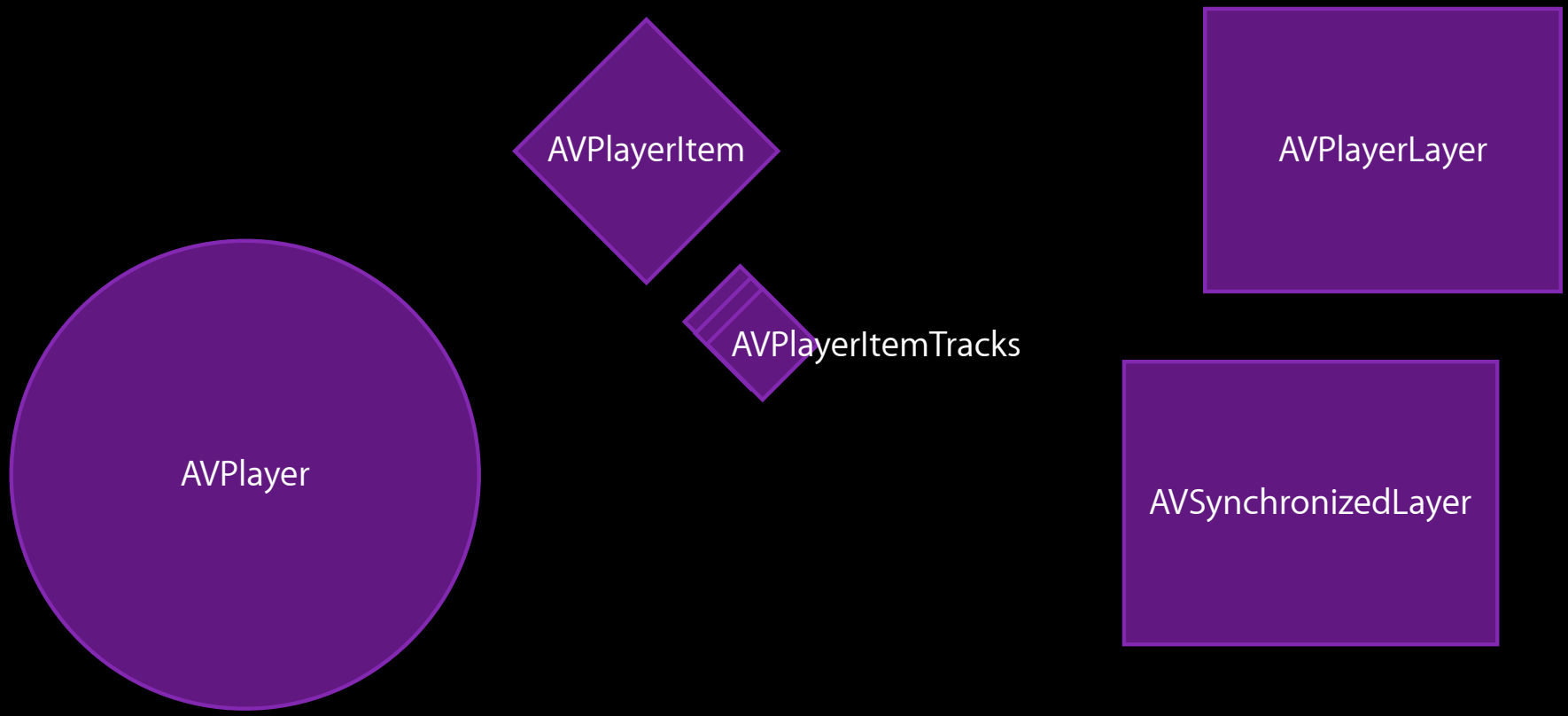
AV Foundation Classes by Function

Media inspection and representation



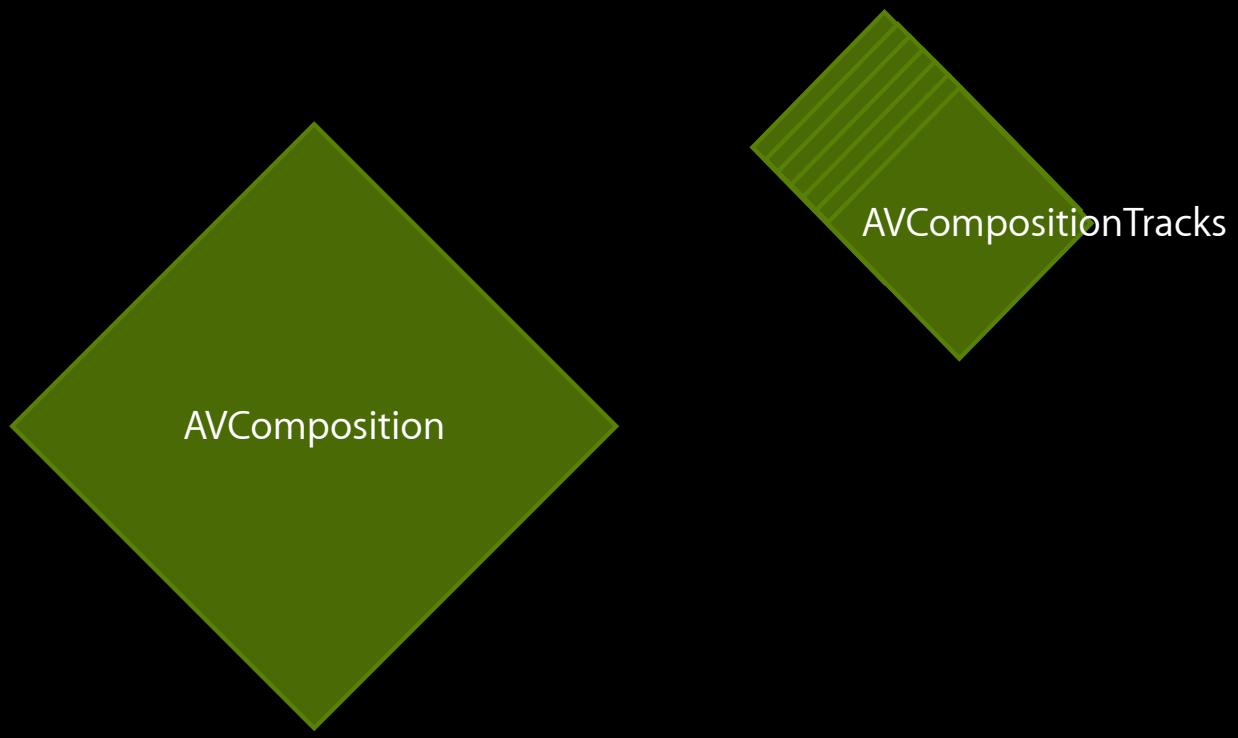
AV Foundation Classes by Function

Audiovisual playback



AV Foundation Classes by Function

Editing



AV Foundation Classes by Function

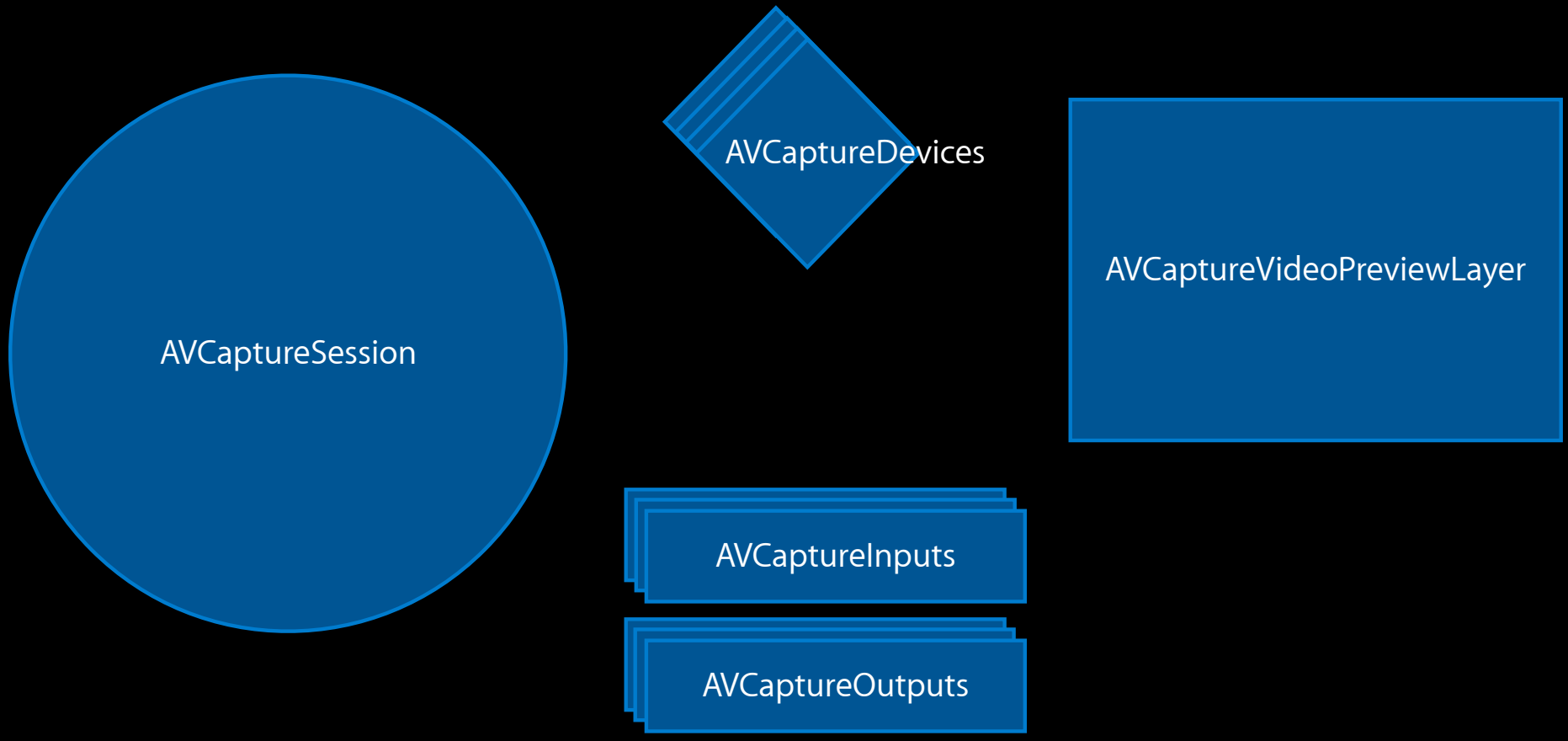
Editing

AV Foundation Classes by Function

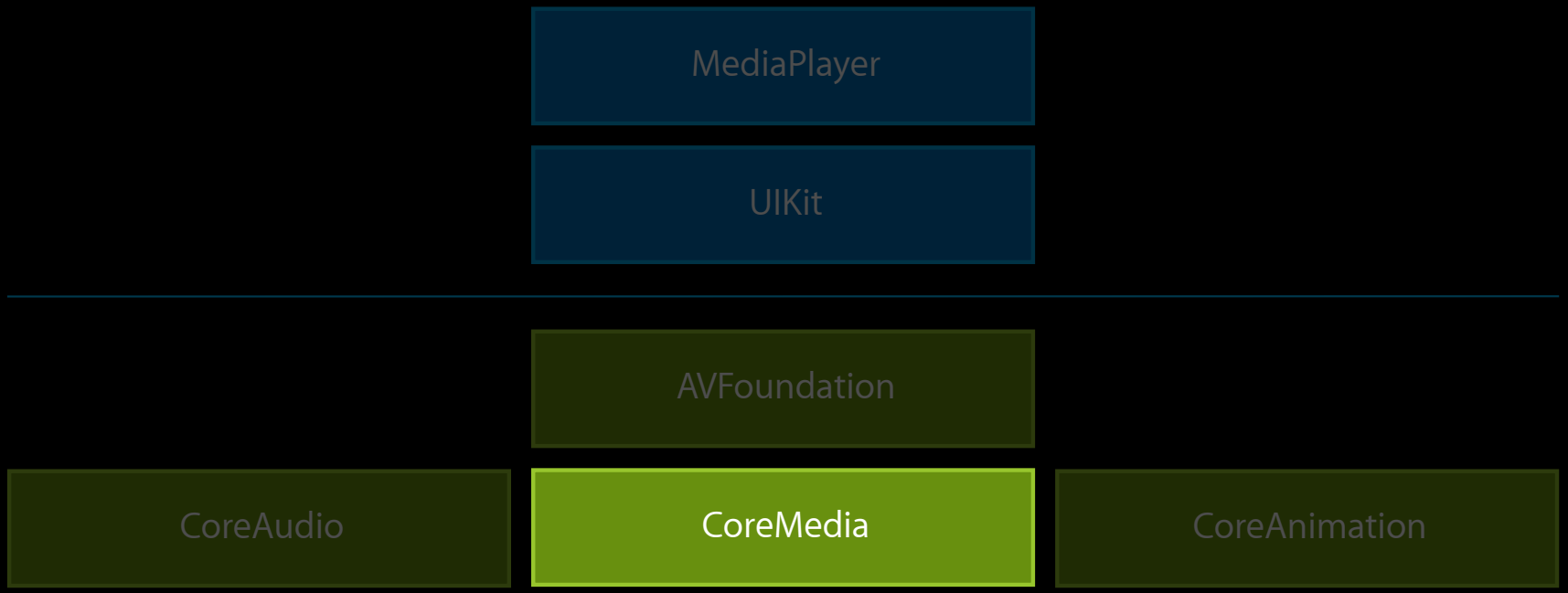
Re-encoding and export

AV Foundation Classes by Function

Access to camera and media capture



Fundamentals



Core Media

- Lower-level still
- Provides many of the primitives used in AV Foundation
  - Time-related data structures
    - CMTime
    - CMTimeRange
    - CMTimeMapping
  - Objects for description and carriage of media data
    - CMFormatDescription
    - CMSampleBuffer

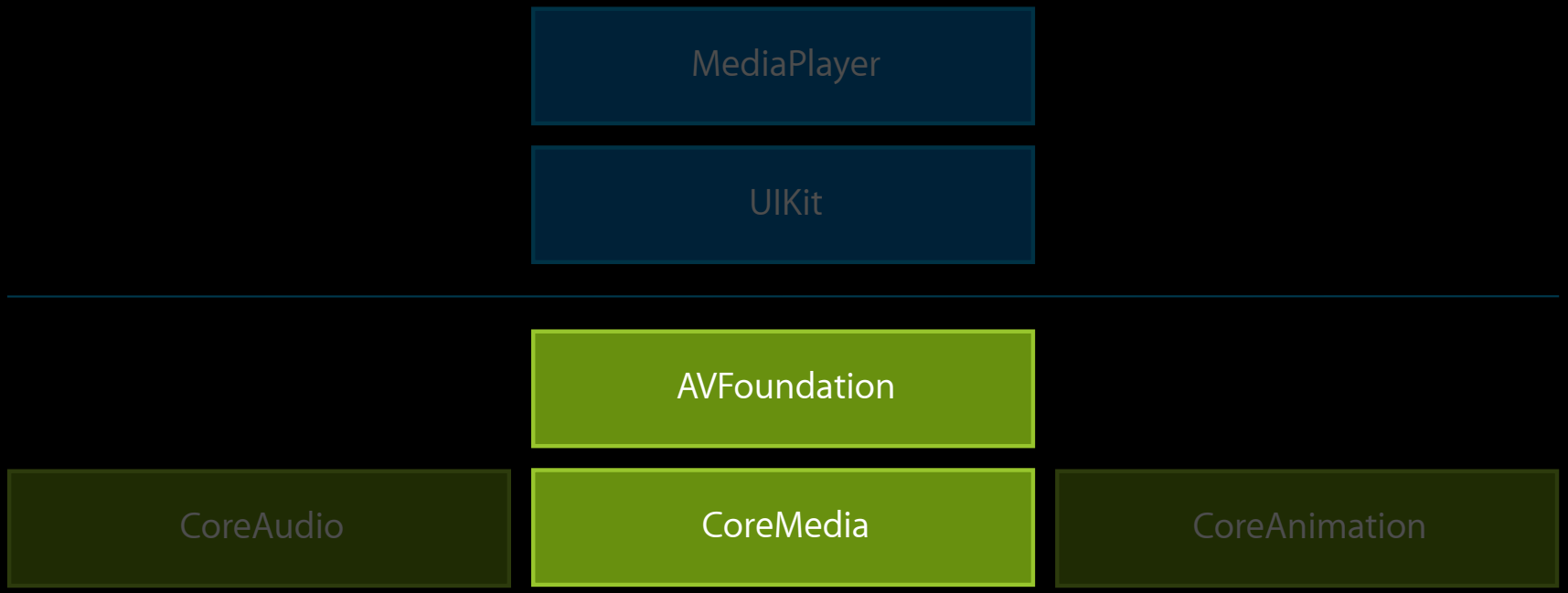
CMTime

- CMTime CMTimeMake(int64_t value, int32_t timescale);
- CMTime CMTimeMakeWithSeconds(Float64 seconds,
  int32_t preferredTimeScale);
- Constants for special values
  - kCMTimeZero
  - kCMTimeInvalid
  - kCMTimePositiveInfinity, kCMTimeNegativeInfinity
- Arithmetic
- Comparison

CMTimeRange

- Another C struct
  - Start time
  - Duration
- CMTime CMTimeRangeMake(CMTime start, CMTime duration);
- Constants for special values
  - kCMTimeRangeInvalid
- Containment, unions, and intersections

Moving Up

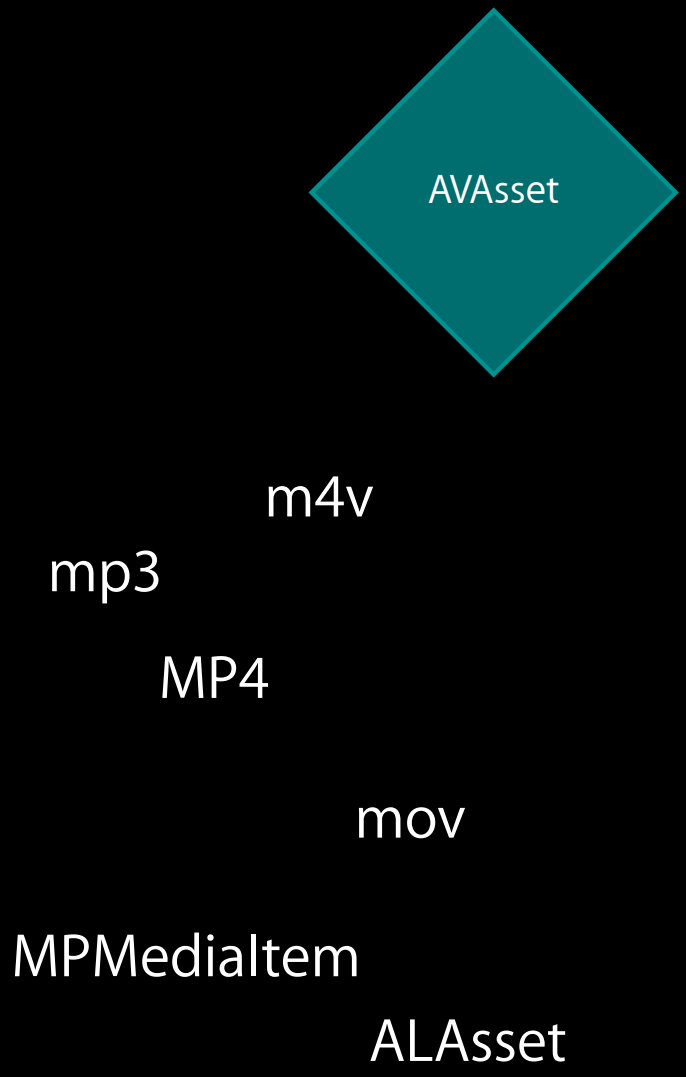

MediaPlayer
UIKit
AVFoundation
CoreAudio
CoreMedia
CoreAnimation

Assets

Representation of timed media

AVAsset Representation

- Model for time-based media resources
  - Provides information about whole resource
    - Duration, title, etc.
  - Presentation hints
    - Natural size
- Can contain one or more streams of media data, each of uniform type
- Not tied to particular data format



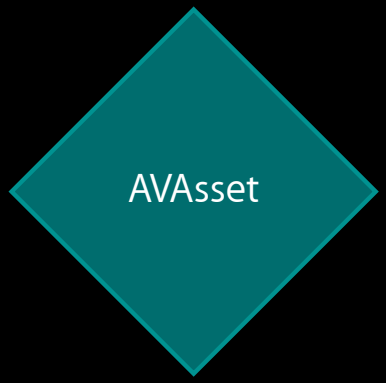
AVAssetTrack: Media Decomposition

- A sequence of media data of uniform type
- Each track has its own
  - Type—e.g., audio or video
  - Format description(s)
  - Timeline
  - Set of one or more segments within its timeline
- A typical asset has a video track and a concomitant audio track

Workflow Example

Create an asset

```
AVURLAsset *asset = [AVURLAsset URLAssetWithURL:url options:opts]
```

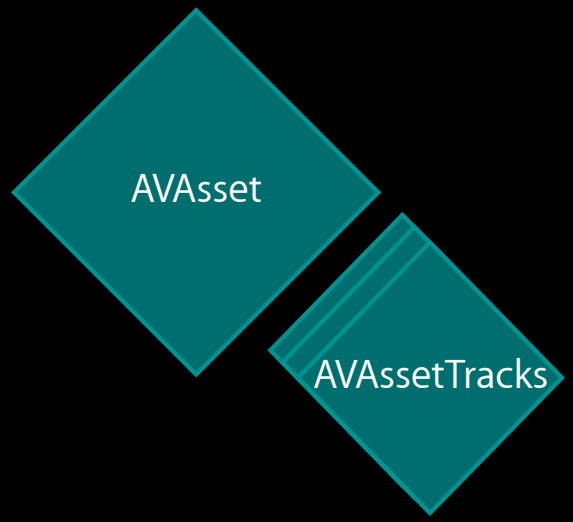


Initialization does not indicate readiness or suitability for any particular purpose…

Getting Asset Information Asynchronously

AVAsynchronousKeyValueLoading protocol

- Determine whether the value for a key is available
- Load absent values



AVAsynchronousKeyValueLoading

Obtaining the status of a value for a key

AVAsynchronousKeyValueLoading

Loading the value for a key

```
AVAsset *asset = [AVURLAsset URLAssetWithURL:myURL options:nil];

NSArray *keys = [NSArray arrayWithObject:@"tracks"];

[asset loadValuesAsynchronouslyForKeys:keys completionHandler:^() {

    NSError *error = nil;
    AVKeyValueStatus tracksStatus = [asset statusOfValueForKey:@"tracks"
    error:&error];
    switch (tracksStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUserInterfaceForTracks];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        case AVKeyValueStatusCancelled:
            break;
   }

}];
```

```
AVAsset *asset = [AVURLAsset URLAssetWithURL:myURL options:nil];

NSArray *keys = [NSArray arrayWithObject:@"tracks"];

[asset loadValuesAsynchronouslyForKeys:keys completionHandler:^() {

    NSError *error = nil;
    AVKeyValueStatus tracksStatus = [asset statusOfValueForKey:@"tracks"
    error:&error];
    switch (tracksStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUserInterfaceForTracks];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        case AVKeyValueStatusCancelled:
            break;
   }

}];
```

```
AVAsset *asset = [AVURLAsset URLAssetWithURL:myURL options:nil];

NSArray *keys = [NSArray arrayWithObject:@"tracks"];

[asset loadValuesAsynchronouslyForKeys:keys completionHandler:^() {

    NSError *error = nil;
    AVKeyValueStatus tracksStatus = [asset statusOfValueForKey:@"tracks"
    error:&error];
    switch (tracksStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUserInterfaceForTracks];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        case AVKeyValueStatusCancelled:
            break;
   }

}];
```

```
AVAsset *asset = [AVURLAsset URLAssetWithURL:myURL options:nil];

NSArray *keys = [NSArray arrayWithObject:@"tracks"];

[asset loadValuesAsynchronouslyForKeys:keys completionHandler:^() {

    NSError *error = nil;
    AVKeyValueStatus tracksStatus = [asset statusOfValueForKey:@"tracks"
    error:&error];
    switch (tracksStatus) {
        case AVKeyValueStatusLoaded:
            [self updateUserInterfaceForTracks];
            break;
        case AVKeyValueStatusFailed:
            [self reportError:error forAsset:asset];
            break;
        case AVKeyValueStatusCancelled:
            break;
   }

}];
```

Inspecting Assets

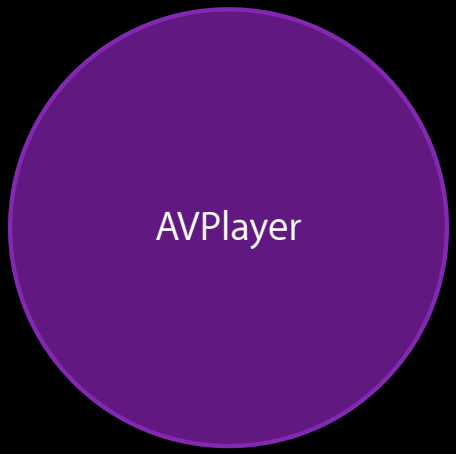
- You want to find out about an asset
  - Duration, title, natural size, etc.
- Getting the answer to a direct question may take time
  - Example: MP3 files may not contain summary information
- You can ask for the work to be done synchronously, but…
  - This may not be ideal
  - What happens if it takes too long?
- Use AVAsynchronousKeyValueLoading methods to ask for information and move on

AVAsset Initialization Option

- Obtaining an exact duration may be an unnecessary expense
- An approximate duration
  - Is often cheaper
  - Is usually sufficient for playback scenarios
- Default behavior does not require precise timing and duration
  - Query AVAsset.providesPreciseDurationAndTiming to determine the accuracy of its duration
- Set AVURLAssetPreferPreciseDurationAndTimingKey in the options dictionary to specify what you want

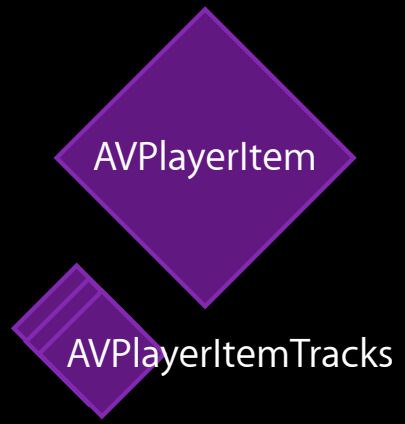
Playback

Playing Media



- Control playback via AVPlayer
  - e.g., rate
- You can manage and observe aspects of playback
- The player plays items
  - Has property currentItem, which may be nil
  - Each item is an instance of AVPlayerItem

Player Items and Tracks



- Presentation state of an asset for playback: AVPlayerItem
- Initialize with AVAsset or NSURL
  - Initializing with an NSURL creates asset
- Control over time
  - "Start time" is the initial setting of currentTime
  - End times for forward and reverse playback
  - Methods for seeking and stepping
- AVPlayerItemTracks wrap the presentation state of corresponding AVAssetTracks
  - Enabled

Workflow Example One

Create and prepare file-based asset for playback

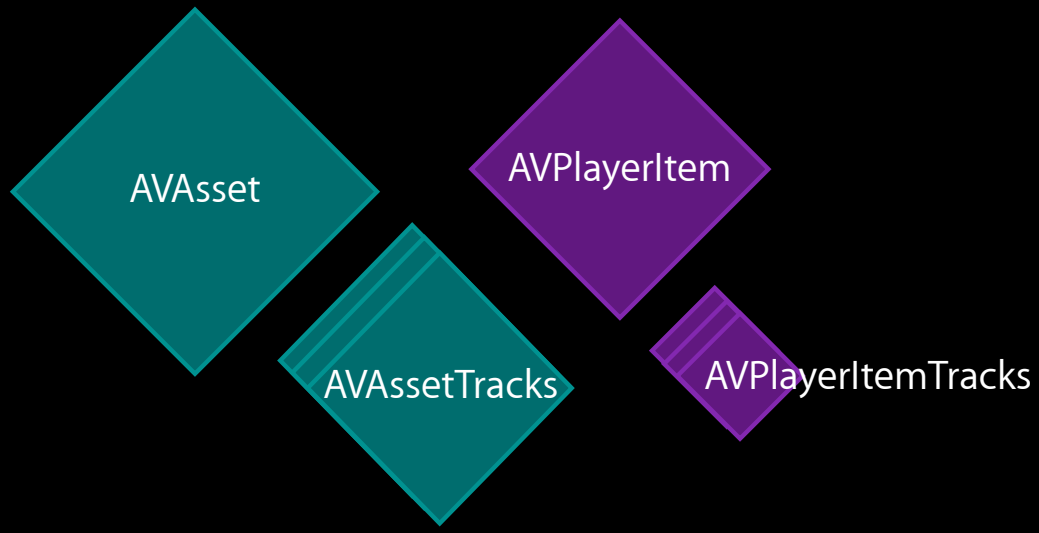
```
AVAsset *asset = [AVURLAsset URLAssetWithURL:myURL options:nil];

[asset loadValuesAsynchronouslyForKeys:[NSArray
arrayWithObject:@”tracks”] completionHandler:myBlock];
```

Workflow Example One

Create an AVPlayerItem with the AVAsset

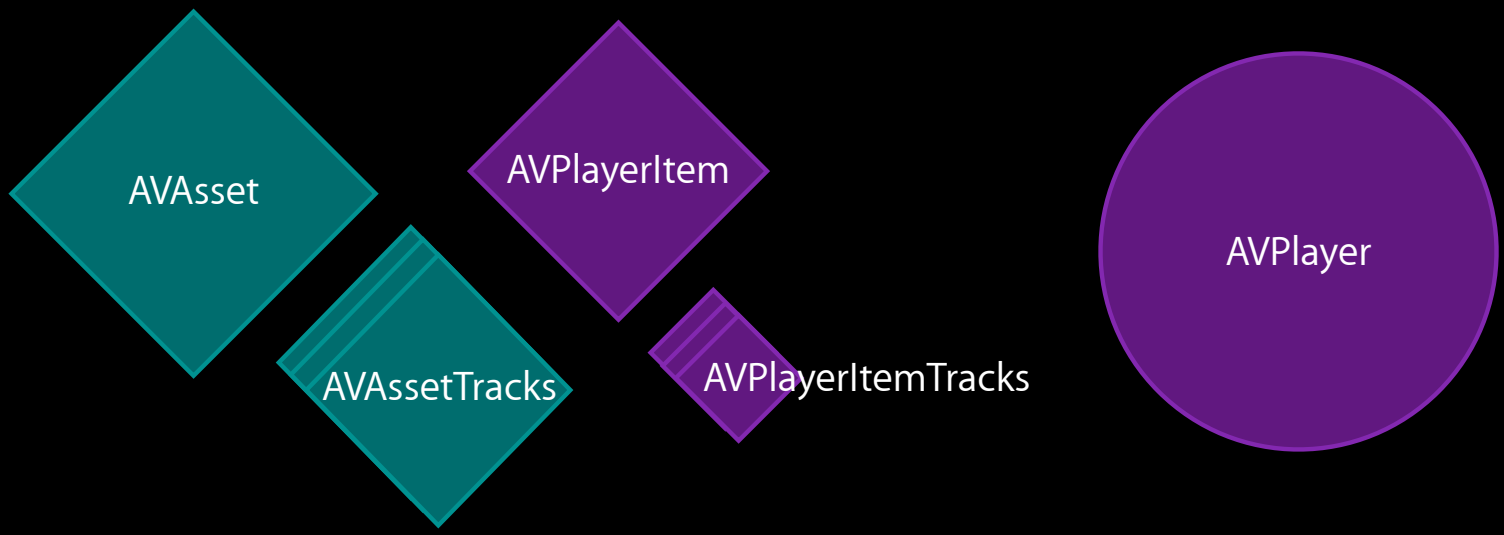
```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithAsset:asset]
[playerItem addObserver:self forKeyPath:@"status" ...]
```



Workflow Example One

Create an AVPlayer with the AVPlayerItem

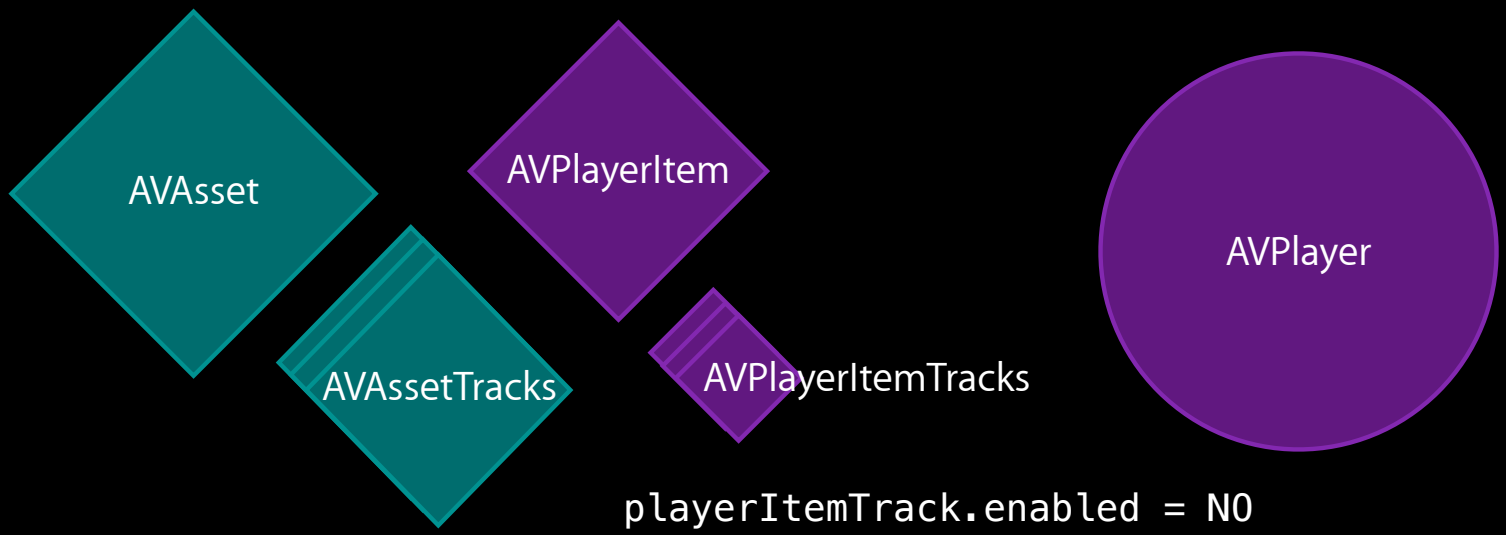
```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem]
```



Workflow Example One

Disable a track for playback

```
if (playerItem.status == AVPlayerItemStatusReadyToPlay)
```

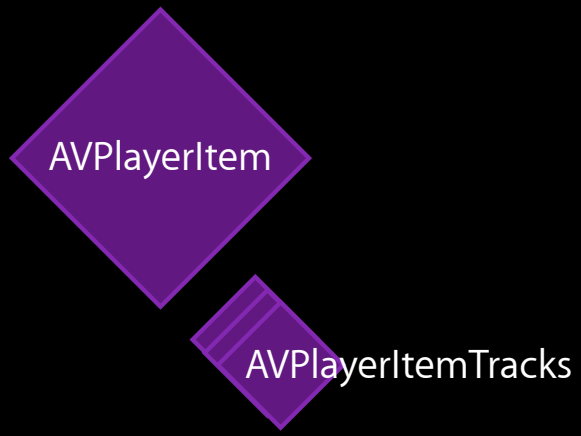



```
playerItemTrack.enabled = NO
[player play]
```

Workflow Example Two

Create and prepare an HTTP live stream for playback

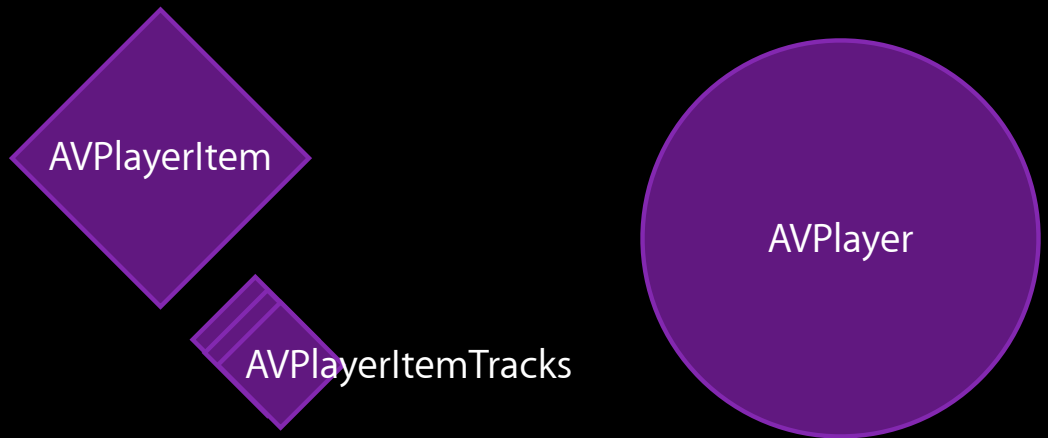
```
AVPlayerItem *playerItem = [AVPlayerItem playerItemWithURL:URL]
[playerItem addObserver:self forKeyPath:@"status" ...]
```



Workflow Example Two

Create an AVPlayer with the AVPlayerItem

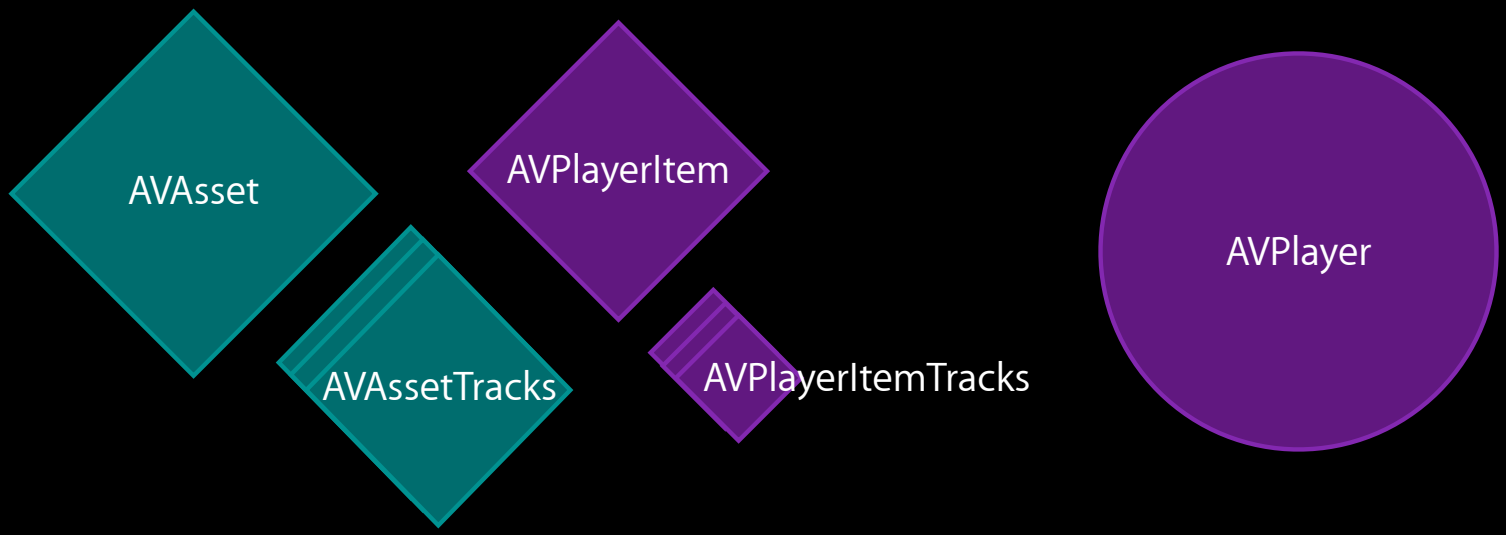
```
AVPlayer *player = [AVPlayer playerWithPlayerItem:playerItem]
```



Workflow Example Two

AVPlayerItem provides an instance of AVAsset for inspection

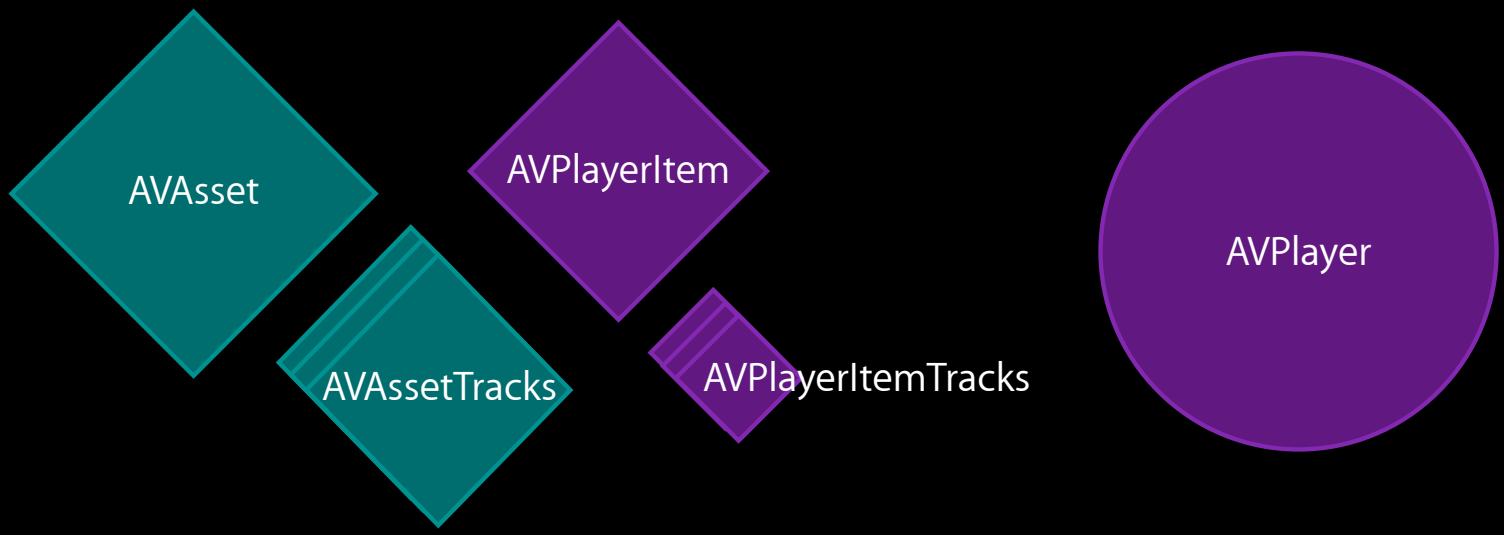
```
if (playerItem.status == AVPlayerItemStatusReadyToPlay)
```



Workflow Example Two

Play

```
if (playerItem.status == AVPlayerItemStatusReadyToPlay)
```




```
[player play]
```

Workflow Example Two (Alternate Take)

Just create the AVPlayer with the HTTP live stream URL

```
AVPlayer *player = [AVPlayer playerWithURL:streamURL]
```

- Does not become ready synchronously
  - Observe AVPlayer.status to determine readiness
- Creates an AVPlayerItem
  - Observe currentItem to get access to it

Workflow Example Three

If you don't know what kind of URL you have

- Try it as a file first, then as a live stream
  - Initialize an AVURLAsset, load its tracks key
    - Success: Proceed with file-based workflow
      - AVPlayerItem with AVURLAsset etc.
    - Failure: Proceed with live stream workflow
      - AVPlayerItem with NSURL etc.
- Code paths converge when the AVPlayerItem reports that it's ready to play

Control over Item Time

- `-[AVPlayerItem seekToTime:]` is tuned for performance
  - Not necessarily precise
- To control precision of seeking, use
  - `-seekToTime:toleranceBefore:toleranceAfter:`
  - A tolerance of 0 can be expensive

Sources of Media

Playback from Camera Rolls

AssetsLibrary.framework

- Start with ALAssetsLibrary, enumerate groups of desired type
- Enumerate assets of each desired group
- Once you have a specific ALAsset, get its URL
- Proceed with file-based playback workflow

Playback from iPod Library

MediaPlayer.framework

- Find item to play via MPMediaQuery
- Once you have a specific MPMediaItem, get its URL
  - `MPMediaItemPropertyAssetURL`
- Proceed with file-based playback workflow

Tracking State, Staying in Time

Tracking Presentation State

- Key-value observing is your friend
- You observe both your own changes and those initiated by the framework
  - AVPlayer rate may be changed when the user multitasks
  - AVPlayerItem loadedTimeRanges and seekableTimeRanges are updated as remote media is transferred
  - AVPlayerItem tracks can be updated while playing an HTTP live stream
  - Media services may be reset
- Register and unregister for key-value observation on main thread

Tracking Readiness for Visual Display

- Observe `AVPlayerLayer.readyForDisplay`
  - Indicates whether the AVPlayerLayer has user-visible content
  - Especially useful for CALayer animations

Tracking Time

`-[AVPlayer addPeriodicTimeObserverForInterval:queue:usingBlock:]`

- Block invoked periodically at the interval requested
- Also when time jumps
- Also when playback starts or stops

`-[AVPlayer addBoundaryTimeObserverForTimes:queue:usingBlock:]`

- Block invoked after a boundary time is traversed

Responding to Completion of Item Playback

```
AVPlayerItemDidPlayToEndTimeNotification
```

Platform Citizenship

Audio Set-Up

- AVAudioSesssion is your friend
  - Typical audio session category: AVAudioSessionCategoryPlayback
  - Supply a delegate that implements
    - `-beginInterruption`
    - `-endInterruptionWithFlags:`

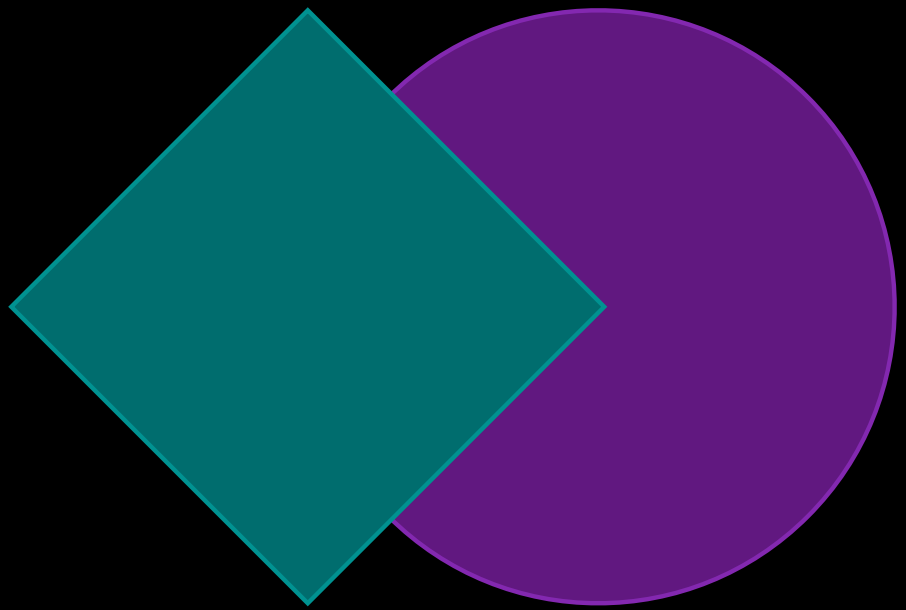
Multitasking

- AVPlayerItems with video are automatically paused when an app goes to the background
  - AVPlayerLayer manages the disengagement and reengagement of visual output of the AVPlayer
  - If the app is set up to get processing time in the background, it can choose to continue playing the audio portion of an item there
- AVAsset key-value loading is not perturbed by multitasking switches

Review

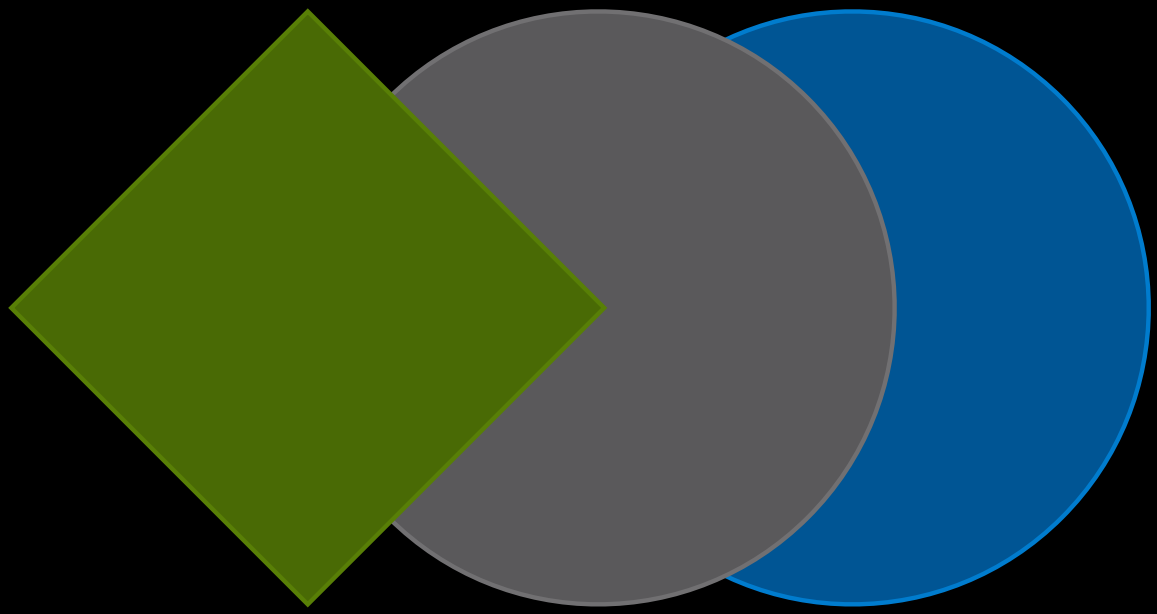
Classes and Functionality

- AV Foundation overview
- Inspection
- Playback



But There's More

- Editing
- Export
- Control over camera and other input devices



New Programming Patterns

- Asynchronous key-value loading
  - Allows you to ask questions about state without blocking the current thread
    - Recall 'A simple question… may take a long time to answer'
- Key-value observing
  - Allows you to monitor changes to an object's properties
- Using blocks as callbacks
  - Simplifies program structure
    - You determine the contextual information you require

Wrap-Up

More Information

Eryk Vershen
Media Technologies Evangelist
eversheN@apple.com

Documentation
AV Foundation Framework Reference
http://developer.apple.com/iphone

Apple Developer Forums
http://devforums.apple.com

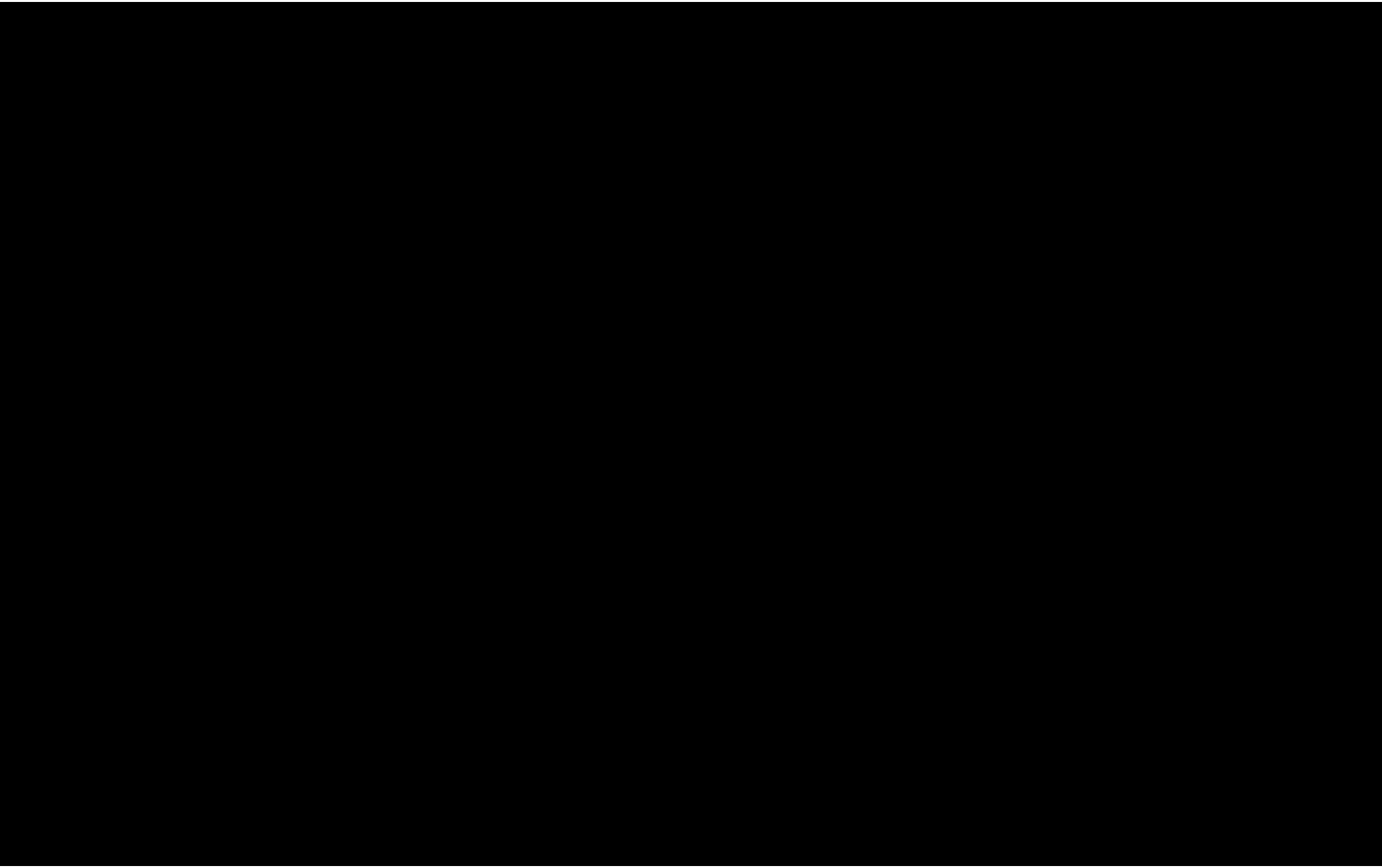
Related Sessions

Editing Media with AV Foundation	Presidio Tuesday 3:15PM
Using the Camera with AV Foundation	Presidio Tuesday 4:30PM
Fundamentals of Digital Audio for Mac OS X and iPhone OS	Mission Wednesday 10:15AM
Discovering AV Foundation (Repeat)	Nob Hill Thursday 4:30PM
Introducing Blocks and Grand Central Dispatch on iPhone	Russian Hill Wednesday 11:30AM
Simplifying iPhone App Development with Grand Central Dispatch	Mission Friday 10:15AM

Labs

AV Foundation Lab #1	Graphics and Media Lab C Wednesday 9:00AM-1:45PM
AV Foundation Lab #2	Graphics and Media Lab B Friday 11:30AM-2:30PM

Summary

- Asynchrony is normative
- Successful initialization does not necessarily indicate readiness or even fitness for a particular purpose
  - With object in hand
    - Prepare asynchronously, where appropriate
    - Operate asynchronously, where appropriate
  - Observe object status
  - Be prepared for completions to occur “at some later time”
- Stay current, stay responsive, stay in time, stay alive
- Stay tuned

WWDC10